AMB Access Bitcoin

Developer Guide

AMB Access Bitcoin: Developer Guide

Table of Contents

What is Amazon Managed Blockchain (AMB) Access Bitcoin?

Amazon Managed Blockchain (AMB) Access provides you with public blockchain nodes for Ethereum and Bitcoin, and you can also create private blockchain networks with the Hyperledger Fabric framework. Choose from various methods to engage with public blockchains, including fully managed, single-tenant (dedicated), and serverless multi-tenant API operations to public blockchain nodes. For use cases where access controls are important, you can choose from fully managed private blockchain networks. Standardized API operations give you instant scalability on a fully managed, resilient infrastructure, so you can build blockchain applications.

AMB Access gives you two distinct types of blockchain infrastructure services: multi-tenant blockchain network access API operations and dedicated blockchain nodes and networks. With dedicated blockchain infrastructure, you can create and use public Ethereum blockchain nodes and private Hyperledger Fabric blockchain networks for your own use. Multi-tenant, API-based offerings, however, such as AMB Access Bitcoin, are composed of a fleet of Bitcoin nodes behind an API layer where the underlying blockchain node infrastructure is shared among customers.

Bitcoin is a decentralized blockchain network that enables secure peer-to-peer transactions of value denominated in the network's native cryptocurrency, Bitcoin (BTC). The Bitcoin network is used by individuals, financial institutions, fintech companies, governments, and more. The Bitcoin network is a medium of exchange, a commodity for investment, or a publicly verifiable and immutable ledger for inscribed data. With Amazon Managed Blockchain (AMB) Access Bitcoin, you can access a pool of Bitcoin Mainnet and Testnet networks through Regional endpoints, through which you can write transactions, read data from the ledger, and invoke JSON-RPC requests available on the Bitcoin Core node client. With serverless Bitcoin endpoints, you can focus on building your applications instead of investing in undifferentiated work such as provisioning, maintaining, and load-balancing Bitcoin nodes. Whether you're building a Bitcoin wallet, building a crypto exchange, or analyzing Bitcoin blockchain data, you only pay for the requests that you make through the Bitcoin endpoints by using AMB Access Bitcoin.

Are you a first-time AMB Access Bitcoin user?

If you are a first-time user of AMB Access Bitcoin, we recommend that you begin by reading the following sections:

- Key concepts: Amazon Managed Blockchain (AMB) Access Bitcoin (p. 2)
- Getting started with Amazon Managed Blockchain (AMB) Access Bitcoin (p. 6)
- Bitcoin use cases with Amazon Managed Blockchain (AMB) Access Bitcoin (p. 11)
- Supported Bitcoin JSON-RPCs with Amazon Managed Blockchain (AMB) Access Bitcoin (p. 13)

Key concepts: Amazon Managed Blockchain (AMB) Access Bitcoin

> **Note**
> This guide assumes that you're familiar with the concepts that are essential to Bitcoin. These concepts include decentralization, nodes, transactions, proof-of-work, wallets, public and private keys, halvings, and others. Before using Amazon Managed Blockchain (AMB) Access Bitcoin, we recommend that you review the Bitcoin Development Documentation and Mastering Bitcoin.

Amazon Managed Blockchain (AMB) Access Bitcoin provides you with serverless access to the Bitcoin blockchain, without requiring you to provision and manage any Bitcoin infrastructure, including nodes. You can use this managed service to access the Bitcoin networks quickly and on-demand, reducing your overall cost of ownership.

The AMB Access Bitcoin provides you with access to the Bitcoin network through full nodes running the Bitcoin Core client, with the wallet functionality disabled, and supporting several JSON Remote Procedure (JSON-RPC) calls. You can invoke Bitcoin JSON RPCs to communicate with Bitcoin nodes managed by Managed Blockchain to interact with the Bitcoin networks. With the Bitcoin JSON-RPCs, you can read data and write transactions, including querying data and submitting transactions to the Bitcoin networks by using the Amazon Managed Blockchain service.

> **Important**
> You are responsible for creating, maintaining, using, and managing your Bitcoin addresses. You are also responsible for the contents of your Bitcoin addresses. AWS is not responsible for any transactions deployed or called using Bitcoin nodes on Amazon Managed Blockchain.

Considerations and limitations for using Amazon Managed Blockchain (AMB) Access Bitcoin

- **Supported Bitcoin networks**

  AMB Access Bitcoin supports the following public networks:
  - **Mainnet**—The public Bitcoin blockchain secured by proof-of-work consensus, and on which the Bitcoin (BTC) cryptocurrency is issued and transacted. Transactions on Mainnet have actual value (that is, they incur real costs) and are recorded on the public blockchain.
  - **Testnet**—The testnet is an alternative Bitcoin blockchain used for testing. Testnet coins are separate and distinct from actual Bitcoin (BTC) and do not usually have any value.

    > **Note**
    > Private networks are not supported.
- **Supported Regions**

  The following are the supported Regions for this service:

Region name	Code	Region
US East (N. Virginia)	IAD	`us-east-1`

Region name	Code	Region
Asia Pacific (Tokyo)	NRT	ap-northeast-1
Asia Pacific (Seoul)	ICN	ap-northeast-2
Asia Pacific (Singapore)	SIN	ap-southeast-1
Europe (Ireland)	DUB	eu-west-1
Europe (London)	LHR	eu-west-2

- **Service endpoints**

  The following are the service endpoints for AMB Access Bitcoin. To connect with the service, you must use an endpoint that includes one of the supported Regions.
  - `mainnet.bitcoin.managedblockchain.`*`Region`*`.amazonaws.com`
  - `testnet.bitcoin.managedblockchain.`*`Region`*`.amazonaws.com`

  For example: `mainnet.bitcoin.managedblockchain.eu-west-2.amazonaws.com`
- **Mining not supported**

  AMB Access Bitcoin does not support Bitcoin (BTC) mining.
- **Signature Version 4 signing of Bitcoin JSON-RPC calls**

  When making calls to the Bitcoin JSON-RPCs on Amazon Managed Blockchain, you can do so over an HTTPS connection authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account can make Bitcoin JSON-RPC calls. To do this, AWS credentials (an access key ID and a secret access key) must be provided with the call.

  Important
  - Do not embed client credentials in user-facing applications.
  - You cannot use IAM policies to restrict access to individual Bitcoin JSON-RPCs.
- **Only submissions of raw transactions are supported**

  Use the `sendrawtransaction` JSON-RPC to submit transactions that update the Bitcoin blockchain state.
- **Request quotas**

  AMB Access Bitcoin has a default quota of 100 requests per second (RPS), per network, per Region for making requests to the Bitcoin Mainnet and Testnet networks. Contact AWS to increase this limit.
- **AWS CloudTrail logging support**

  You can configure CloudTrail to log your Bitcoin JSON-RPCs. For more information, see Logging Amazon Managed Blockchain (AMB) Access Bitcoin events by using AWS CloudTrail (p. 31)

Setting up Amazon Managed Blockchain (AMB) Access Bitcoin

Before you use Amazon Managed Blockchain (AMB) Access Bitcoin for the first time, follow the steps in this section to create an AWS account. The following chapter discusses how to start using AMB Access Bitcoin.

Prerequisites and considerations

Before you use AWS for the first time, you must have an AWS account.

Sign up for AWS

When you sign up for AWS, your AWS account is automatically signed up for all AWS services, including Amazon Managed Blockchain (AMB) Access Bitcoin. You're charged only for the services that you use.

If you have an AWS account already, go to the next step. If you don't have an AWS account, use the following procedure to create one.

To create an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to an administrative user, and use only the root user to perform tasks that require root user access.

Create an IAM user with appropriate permissions

To create and work with AMB Access Bitcoin, you must have an AWS Identity and Access Management (IAM) principal (user or group) with permissions that allow necessary Managed Blockchain actions.

Only IAM principals can make Bitcoin JSON-RPC calls. When making calls to the Bitcoin JSON-RPCs on Amazon Managed Blockchain, you can do so over an HTTPS connection authenticated using the Signature Version 4 signing process. This means that only authorized IAM principals in the AWS account can make Bitcoin JSON-RPC calls. To do this, AWS credentials (an access key ID and a secret access key) must be provided with the call.

For information about how to create an IAM user, see Creating an IAM user in your AWS account. For more information about how to attach a permissions policy to a user, see Changing permissions for an IAM user. For an example of a permissions policy that you can use to give a user permission to work with AMB Access Bitcoin, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Bitcoin (p. 26).

Install and configure the AWS Command Line Interface

If you have not already done so, install the latest AWS Command-Line Interface (CLI) to work with AWS resources from a terminal. For more information, see Installing or updating the latest version of the AWS CLI.

Note
For CLI access, you need an access key ID and a secret access key. Use temporary credentials instead of long-term access keys when possible. Temporary credentials include an access key ID, a secret access key, and a security token that indicates when the credentials expire. For more information, see Using temporary credentials with AWS resources in the *IAM User Guide*.

Getting started with Amazon Managed Blockchain (AMB) Access Bitcoin

Use the step-by-step tutorials in this section to learn how to perform tasks by using Amazon Managed Blockchain (AMB) Access Bitcoin. These examples require you to complete some prerequisites. If you are new to AMB Access Bitcoin, review the *Setting up* section of this guide to make sure you have completed those prerequisites. For more information, see Setting up Amazon Managed Blockchain (AMB) Access Bitcoin (p. 4).

Topics

Create an IAM policy to access Bitcoin JSON-RPCs

In order to access the public endpoints for the Bitcoin Mainnet and Testnet to make JSON-RPC calls, you must have user credentials (AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY) that have the appropriate IAM permissions for Amazon Managed Blockchain (AMB) Access Bitcoin. In a terminal with the AWS CLI installed, run the following command to create an IAM Policy to access both Bitcoin endpoints:

```
cat <<EOT > ~/amb-btc-access-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid" : "AMBBitcoinAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:InvokeRpcBitcoin*"
            ],
            "Resource": "*"
        }
    ]
}
EOT
aws iam create-policy --policy-name AmazonManagedBlockchainBitcoinAccess --policy-document
 file://$HOME/amb-btc-access-policy.json
```

Note
The previous example gives you to access both Mainnet and Testnet. To get access to a specific endpoint, use the following `Action` command:

- `"managedblockchain:InvokeRpcBitcoinMainnet"`
- `"managedblockchain:InvokeRpcBitcoinTestnet"`

After you create the policy, attach that policy to your IAM user's Role for it to take effect. In the AWS Management Console, navigate to the IAM service, and attach the policy `AmazonManagedBlockchainBitcoinAccess` to the Role assigned to your IAM user. For more information, see Creating a Role and assigning to an IAM user.

Make AMB Access Bitcoin JSON-RPC requests in awscurl by using the AWS CLI

Example

Sign requests with your IAM user credentials by using Signature Version 4 (SigV4) in order to make Bitcoin JSON-RPC calls to the AMB Access Bitcoin endpoints. The awscurl command line tool can help you sign requests to AWS services using SigV4. For more information, see the awscurl README.md.

Install awscurl by using the method appropriate to your operating system. On macOS, HomeBrew is the recommended application:

```
brew install awscurl
```

If you have already installed and configured the AWS CLI, your IAM user credentials and default AWS Region are set in your environment and have access to awscurl. Using awscurl, submit a request to both the Bitcoin *Mainnet* and *Testnet* by invoking the `getblock` RPC. This call accepts a string parameter corresponding to the block hash for which you want to retrieve information.

The following command retrieves the block header data from the Bitcoin Mainnet by using the block hash in the `params` array to select the specific block for which to retrieve the headers. This example uses the `us-east-1` endpoint. You can replace this with your preferred Bitcoin JSON-RPC and AWS Region that is supported by Amazon Managed Blockchain (AMB) Access Bitcoin. Furthermore, you can make a request against the Testnet network, rather than Mainnet, by replacing `mainnet` with `testnet` in the command.

```
awscurl -X POST -d '{ "jsonrpc": "1.0", "id": "getblockheader-curltest", "method":
 "getblockheader", "params":
 ["000000000000000000105bebab2f9dd16234a30950d38ec6ddc24d466e750a0"] }' --service
 managedblockchain https://mainnet.bitcoin.managedblockchain.us-east-1.amazonaws.com  --
region us-east-1 -k
```

The results include details from the block headers and a list of transaction hashes included in the requested block. See the following example:

```
{"result":{"hash":"000000000000000000105bebab2f9dd16234a30950d38ec6ddc24d466e750a0",
      "confirmations":2,"height":799243,"version":664485888,"versionHex":"279b4000",
      "merkleroot":"568e79752e1921ecf40c961435abb41bc5700fe2833ecadc4abfc2f615ddc1b8",
      "time":1689684290,"mediantime":1689681317,"nonce":2091174943,"bits":"17053894",
      "difficulty":53911173001054.59,
      "chainwork":"0000000000000000000000000000000000000000004f375cf72ff64e2404c1589c",
      "nTx":2135,

 "previousblockhash":"00000000000000000002ffe4efe07ae74ec8b92c7696f5e12b5da506f015ba6b",
      "nextblockhash":"00000000000000000038f05ddcf3f483fdcb74f4be606c022bcb673424fa4ca"},
```

```
    "error":null,"id":"curltest"}
```

Make Bitcoin JSON-RPC requests in Node.js

You can submit signed requests by using HTTPS to access the Bitcoin *Mainnet* and *Testnet* endpoints and to make JSON-RPC API calls by using the native https module in Node.js, or you can use a third-party library such as AXIOS. The following example shows you how to make a Bitcoin JSON-RPC request to the AMB Access Bitcoin endpoints.

Example

To run this example Node.js script, apply the following prerequisites:

1. You must have node version manager (nvm) and Node.js installed on your machine. You can find installation instructions for your OS here.
2. Use the `node --version` command and confirm that you are using *Node version 14* or higher. If required, you can use the `nvm install 14` command, followed by the `nvm use 14` command, to install *version 14*.
3. The environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` must contain the credentials that are associated with your account. The environment variables `AMB_HTTP_ENDPOINT` must contain your AMB Access Bitcoin endpoints.

   Export these variables as strings on your client by using the following commands. Replace the highlighted values in the following strings with appropriate values from your IAM user account.

```
export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

After you complete all prerequisites, copy the following `package.json` file and `index.js` script into your local environment by using your editor:

package.json

```
{
  "name": "bitcoin-rpc",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@aws-crypto/sha256-js": "^4.0.0",
    "@aws-sdk/credential-provider-node": "^3.360.0",
    "@aws-sdk/protocol-http": "^3.357.0",
    "@aws-sdk/signature-v4": "^3.357.0",
    "axios": "^1.4.0"
  }
}
```

index.js

```
const axios = require('axios');
```

```
const SHA256 = require('@aws-crypto/sha256-js').Sha256
const defaultProvider = require('@aws-sdk/credential-provider-node').defaultProvider
const HttpRequest = require('@aws-sdk/protocol-http').HttpRequest
const SignatureV4 = require('@aws-sdk/signature-v4').SignatureV4

// define a signer object with AWS service name, credentials, and region
const signer = new SignatureV4({
  credentials: defaultProvider(),
  service: 'managedblockchain',
  region: 'us-east-1',
  sha256: SHA256,
});


const rpcRequest = async () => {

  // create a remote procedure call (RPC) request object definig the method, input params
  let rpc = {
    jsonrpc: "1.0",
    id: "1001",
    method: 'getblock',
    params: ["00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09"]
  }

  //bitcoin endpoint
  let bitcoinURL = 'https://mainnet.bitcoin.managedblockchain.us-east-1.amazonaws.com/';

  // parse the URL into its component parts (e.g. host, path)
  const url = new URL(bitcoinURL);

  // create an HTTP Request object
  const req = new HttpRequest({
    hostname: url.hostname.toString(),
    path: url.pathname.toString(),
    body: JSON.stringify(rpc),
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Accept-Encoding': 'gzip',
      host: url.hostname,
    }
  });


  // use AWS SignatureV4 utility to sign the request, extract headers and body
  const signedRequest = await signer.sign(req, { signingDate: new Date() });

  try {
    //make the request using axios
    const response = await axios({...signedRequest, url: bitcoinURL, data: req.body})

    console.log(response.data)
  } catch (error) {
    console.error('Something went wrong: ', error)
    throw error
  }


}

rpcRequest();
```

The previous sample code uses Axios to make RPC requests to the Bitcoin endpoint, and it signs those requests with the appropriate Signature Version 4 (SigV4) headers by using the official AWS SDK v3 tools. To run the code, open a terminal in the same directory as your files and run the following:

```
npm i
node index.js
```

The result that is generated will resemble the following:

```
{"hash":"00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09","
    confirmations":784126,"height":1000, "version":1,"versionHex":"00000001",
    "merkleroot":"fe28050b93faea61fa88c4c630f0e1f0a1c24d0082dd0e10d369e13212128f33",
    "time":1232346882,
    "mediantime":1232344831,"nonce":2595206198,"bits":"1d00ffff","difficulty":1,
    "chainwork":"0000000000000000000000000000000000000000000000000000000000000003e903e903e9",
    "nTx":1,
    "previousblockhash":"0000000008e647742775a230787d66fdf92c46a48c896bfbc85cdc8acc67e87d",
    "nextblockhash":"00000000a2887344f8db859e372e7e4bc26b23b9de340f725afbf2edb265b4c6",
    "strippedsize":216,"size":216,"weight":864,
    "tx":["fe28050b93faea61fa88c4c630f0e1f0a1c24d0082dd0e10d369e13212128f33"]},
    "error":null,"id":"1001"}
```

Note
The sample request in the previous script makes the `getblock` call with the same input parameter block hash as the Make AMB Access Bitcoin JSON-RPC requests in awscurl by using the AWS CLI (p. 7) example. To make other calls, modify the `rpc` object in the script with a different Bitcoin JSON-RPC. You can change the host property option to the Bitcoin `testnet` to make calls on that endpoint.

Make Bitcoin JSON-RPC requests over AWS PrivateLink

AWS PrivateLink is a highly available, scalable technology that you can use to connect your VPC to services privately as if they were in your VPC. You do not have to use an internet gateway, NAT device, public IP address, AWS Direct Connect connection, or AWS Site-to-Site VPN connection to communicate with the service from your private subnets. For more information about AWS PrivateLink or to set up AWS PrivateLink, see What is AWS PrivateLink?

You can access Bitcoin JSON-RPCs over AWS PrivateLink by using a VPC endpoint. Requests to this private endpoint are not passed over the open internet, so you can send requests directly to the Bitcoin endpoints by using the same *SigV4* authentication. Navigate to the Amazon Virtual Private Cloud service in the AWS Management Console, and select “Endpoints” from the options to find your AWS PrivateLink endpoint. Your endpoint will be in the following format: `com.amazonaws.`*`AWS-REGION`*`.managedblockchain.bitcoin.`*`NETWORK-TYPE`*; for example, `com.amazonaws.`*`us-east-1`*`.managedblockchain.bitcoin.`*`testnet`*.

Bitcoin use cases with Amazon Managed Blockchain (AMB) Access Bitcoin

This topic provides a list AMB Access Bitcoin use cases

Topics

Build a Bitcoin (BTC) wallet to send and receive BTC

BTC, the native cryptocurrency on the Bitcoin network, serves as an essential component of the network's security model. It also acts as a commodity and medium of exchange, widely used by institutions, businesses, and individuals. Consequently, many wallet applications rely on Bitcoin nodes to interact with the Bitcoin blockchain. These applications calculate the balance of unspent outputs (UTXOs) for a given set of addresses, sign and send transactions to the Bitcoin network, and retrieve data about historical transactions.

The following is a sample of some of the Bitcoin JSON-RPCs that Amazon Managed Blockchain (AMB) Access Bitcoin supports for BTC wallet transactions:

- `estimatesmartfee`
- `createmultisig`
- `createrawtransaction`
- `sendrawtransaction`

For more information, see Supported JSON-RPCs (p. 13).

Analyze activity on the Bitcoin blockchain

You can analyze the volume of transaction activity on the Bitcoin blockchain by using the `getchaintxstats` JSON-RPC method. This JSON-RPC allows you to access metrics such as average transaction rates per second, total transaction count, block count, and more. You can also define a window of block numbers or a block hash as a delimiter to calculate these statistics for a specific set of blocks in the network, if desired.

For more information, see Supported JSON-RPCs (p. 13).

Verify messages signed using a Bitcoin key pair

Bitcoin wallets have a private key and a public key that make up a key pair. These keys are used to sign transactions and serve as the user's identity on the blockchain. The public key is used to create addresses, which are standardized alphanumeric identifiers (27 to 34 characters long). These addresses are used to receive BTC outputs and handle transactions or messages.

With a Bitcoin wallet, users can also sign and verify messages cryptographically. This process is often used to prove ownership of a specific wallet address and the BTC associated with it. By using the `verifymessage` Bitcoin JSON-RPC, you can check the authenticity and validity of a message signed by another wallet. Specifically, a Bitcoin node can be used to verify if a message has been signed using the private key corresponding to the provided public key derived address within the signed message itself.

For more information, see Supported JSON-RPCs (p. 13).

Inspect the Bitcoin mempool

Many applications need to access the *mempool* to keep track of pending transactions, get a list of all pending transactions, or find out where a transaction came from. To do this, there are Bitcoin JSON-RPCs like `getmempoolancestors`, `getmempoolentry`, and `getrawmempool` that support this activity. These Bitcoin JSON-RPCs help applications get the information they need from the *mempool*.

Amazon Managed Blockchain (AMB) Access Bitcoin also supports the `testmempoolaccept` Bitcoin JSON-RPCs, which allows you to verify if a transaction meets protocol rules and would be accepted by a node before submitting. Wallets, exchanges, and any other entities who directly submit transactions to the Bitcoin blockchain utilize these Bitcoin JSON-RPCs.

For more information, see Supported JSON-RPCs (p. 13).

Supported Bitcoin JSON-RPCs with Amazon Managed Blockchain (AMB) Access Bitcoin

This topic provides a list of and references to the Bitcoin JSON-RPCs that Managed Blockchain supports. Each supported JSON-RPC has a brief description of its use.

Note

- You can authenticate Bitcoin JSON-RPCs on Managed Blockchain by using the Signature Version 4 (SigV4) signing process. This means that only authorized IAM principals in the AWS account can interact with it by using the Bitcoin JSON-RPCs. Provide AWS credentials (an access key ID and secret access key) with the call.
- If your HTTP response is larger than 10 MB, you will get an error. To correct this, you must set the compression headers to `Accept- Encoding:gzip`. The compressed response your client then receives contains the following headers: `Content-Type: application/json` and `Content-Encoding: gzip`.
- Amazon Managed Blockchain (AMB) Access Bitcoin generates a 400 error for malformed JSON-RPC requests.
- Use the `sendrawtransaction` JSON-RPC to submit transactions that update the Bitcoin blockchain state.
- AMB Access Bitcoin has a default request limit of 100 requests per second (RPS), per `NETWORK_TYPE`, per AWS Region. You must create a service limit increase case using the AWS Support Center Console to justify an exception to this limit .

Supported JSON-RPCs

AMB Access Bitcoin supports the following Bitcoin JSON-RPCs. Each supported call has a brief description of its use.

Category	JSON-RPC	Description
Blockchain RPCs	getbestblockhash	Returns the hash of the best (tip) block in the most-work, fully validated chain.
	getblock	If verbosity is 0, returns a string that is serialized, hex-encoded data for block 'hash'. If verbosity is 1, returns an Object with information about block 'hash'. If verbosity is 2, returns an Object with information about block 'hash' and information about each transaction.
	getblockchaininfo	Returns an object containing various state info regarding blockchain processing.
	getblockcount	Returns the height of the most-work, fully validated chain. The genesis block has height 0.

Category	JSON-RPC	Description
	getblockfilter	Retrieves a BIP 157 content filter for a particular block using the block hash.
	getblockhash	Returns hash of block in best-block-chain at height provided.
	getblockheader	If verbose is false, returns a string that is serialized, hex-encoded data for blockheader 'hash'. If verbose is true, returns an Object with information about blockheader 'hash'.
	getblockstats	Computes per block statistics for a given window. All amounts are in satoshis. It won't work for some heights with pruning.
	getchaintips	Returns information about all known tips in the block tree, including the main chain and orphaned branches.
	getchaintxstats	Computes statistics about the total number and rate of transactions in the chain.
	getdifficulty	Returns the proof-of-work difficulty as a multiple of the minimum difficulty.
	getmempoolancestors	If txid is in the mempool, returns all in-mempool ancestors.
	getmempooldescendants	If txid is in the mempool, returns all in-mempool descendants.
	getmempoolentry	Returns mempool data for given transaction.
	getmempoolinfo	Returns details on the active state of the TX memory pool.
	getrawmempool	Returns all transaction IDs in memory pool as a JSON array of string transaction IDs. **Note** `verbose = true` is not supported.
	gettxout	Returns details about an unspent transaction output.
	gettxoutproof	Returns a hex-encoded proof that "txid" was included in a block.
Rawtransactions RPCs	createrawtransaction	Creates a transaction spending the given inputs and creating new outputs.
	decoderawtransaction	Returns a JSON object representing the serialized, hex-encoded transaction.
	decodescript	Decodes a hex-encoded script.
	getrawtransaction	Returns the raw transaction data.

Category	JSON-RPC	Description
	sendrawtransaction	Submits a raw transaction (serialized, hex-encoded) to local node and network.
	testmempoolaccept	Returns result of mempool acceptance tests indicating if raw transaction (serialized, hex-encoded) would be accepted by mempool. This checks if the transaction violates the consensus or policy rules.
Util RPCs	createmultisig	Creates a multi-signature address with *n* signature of *m* keys required.
	estimatesmartfee	Estimates the approximate fee per kilobyte required for a transaction to begin confirmation within conf_target blocks, if possible, and returns the number of blocks for which the estimate is valid. Uses virtual transaction size, as defined in BIP 141 (witness data is discounted).
	validateaddress	Returns information about the given bitcoin address.
	verifymessage	Verifies a signed message.

Security in Amazon Managed Blockchain (AMB) Access Bitcoin

Cloud security at AWS is of the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as both security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Managed Blockchain (AMB) Access Bitcoin, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors, including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

To provide data protection, authentication, and access control, Amazon Managed Blockchain uses AWS features and the features of the open-source framework running in Managed Blockchain.

This documentation helps you understand how to apply the shared responsibility model when using AMB Access Bitcoin. The following topics show you how to configure AMB Access Bitcoin to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AMB Access Bitcoin resources.

Topics

Data protection in Amazon Managed Blockchain (AMB) Access Bitcoin

The AWS shared responsibility model applies to data protection in Amazon Managed Blockchain (AMB) Access Bitcoin. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center (successor to AWS Single Sign-On) or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.

- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AMB Access Bitcoin or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption

Data encryption helps prevent unauthorized users from reading data from a blockchain network and the associated data storage systems. This includes data that might be intercepted as it travels the network, known as *data in transit*.

Encryption in transit

By default, Managed Blockchain uses an HTTPS/TLS connection to encrypt all the data that's transmitted from a client computer that runs the AWS CLI to AWS service endpoints.

You don't need to do anything to enable the use of HTTPS/TLS. It's always enabled unless you explicitly disable it for an individual AWS CLI command by using the `--no-verify-ssl` command.

Identity and access management for Amazon Managed Blockchain (AMB) Access Bitcoin

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AMB Access Bitcoin resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AMB Access Bitcoin.

Service user – If you use the AMB Access Bitcoin service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AMB Access Bitcoin features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AMB Access Bitcoin, see Troubleshooting Amazon Managed Blockchain (AMB) Access Bitcoin identity and access (p. 29).

Service administrator – If you're in charge of AMB Access Bitcoin resources at your company, you probably have full access to AMB Access Bitcoin. It's your job to determine which AMB Access Bitcoin features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AMB Access Bitcoin, see How Amazon Managed Blockchain (AMB) Access Bitcoin works with IAM (p. 22).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AMB Access Bitcoin. To view example AMB Access Bitcoin identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Bitcoin (p. 26).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (successor to AWS Single Sign-On) (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Multi-factor authentication in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *AWS Account Management Reference Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center (successor to AWS Single Sign-On). You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects

in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.

- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Managed Blockchain (AMB) Access Bitcoin in the *Service Authorization Reference*.
- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that

you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of an entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Managed Blockchain (AMB) Access Bitcoin works with IAM

Before you use IAM to manage access to AMB Access Bitcoin, learn what IAM features are available to use with AMB Access Bitcoin.

IAM features you can use with Amazon Managed Blockchain (AMB) Access Bitcoin

IAM feature	AMB Access Bitcoin support
Identity-based policies (p. 22)	Yes
Resource-based policies (p. 23)	No
Policy actions (p. 23)	Yes
Policy resources (p. 24)	No
Policy condition keys (p. 24)	No
ACLs (p. 25)	No
ABAC (tags in policies) (p. 25)	No
Temporary credentials (p. 25)	No
Principal permissions (p. 26)	No
Service roles (p. 26)	No
Service-linked roles (p. 26)	No

To get a high-level view of how AMB Access Bitcoin and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for AMB Access Bitcoin

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for AMB Access Bitcoin

To view examples of AMB Access Bitcoin identity-based policies, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Bitcoin (p. 26).

Resource-based policies within AMB Access Bitcoin

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for AMB Access Bitcoin

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

To see a list of AMB Access Bitcoin actions, see Actions Defined by Amazon Managed Blockchain (AMB) Access Bitcoin in the *Service Authorization Reference*.

Policy actions in AMB Access Bitcoin use the following prefix before the action:

```
managedblockchain:
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "managedblockchain::action1",
      "managedblockchain::action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `InvokeRpcBitcoin`, include the following action:

```
"Action": "managedblockchain::InvokeRpcBitcoin*"
```

To view examples of AMB Access Bitcoin identity-based policies, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Bitcoin (p. 26).

Policy resources for AMB Access Bitcoin

Supports policy resources	No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. Statements must include either a `Resource` or a `NotResource` element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as *resource-level permissions*.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of AMB Access Bitcoin resource types and their ARNs, see Resources Defined by Amazon Managed Blockchain (AMB) Access Bitcoin in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon Managed Blockchain (AMB) Access Bitcoin .

To view examples of AMB Access Bitcoin identity-based policies, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Bitcoin (p. 26).

Policy condition keys for AMB Access Bitcoin

Supports service-specific policy condition keys	No

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (or `Condition` *block*) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical `AND` operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical `OR` operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the *IAM User Guide*.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of AMB Access Bitcoin condition keys, see Condition Keys for Amazon Managed Blockchain (AMB) Access Bitcoin in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon Managed Blockchain (AMB) Access Bitcoin .

To view examples of AMB Access Bitcoin identity-based policies, see Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Bitcoin (p. 26).

ACLs in AMB Access Bitcoin

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with AMB Access Bitcoin

Supports ABAC (tags in policies)	No

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with AMB Access Bitcoin

Supports temporary credentials	No

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically

create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for AMB Access Bitcoin

Supports principal permissions	No

When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon Managed Blockchain (AMB) Access Bitcoin in the *Service Authorization Reference*.

Service roles for AMB Access Bitcoin

Supports service roles	No

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

Warning
Changing the permissions for a service role might break AMB Access Bitcoin functionality. Edit service roles only when AMB Access Bitcoin provides guidance to do so.

Service-linked roles for AMB Access Bitcoin

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon Managed Blockchain (AMB) Access Bitcoin

By default, users and roles don't have permission to create or modify AMB Access Bitcoin resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

For details about actions and resource types defined by AMB Access Bitcoin, including the format of the ARNs for each of the resource types, see Actions, Resources, and Condition Keys for Amazon Managed Blockchain (AMB) Access Bitcoin in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete AMB Access Bitcoin resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the AMB Access Bitcoin console

To access the Amazon Managed Blockchain (AMB) Access Bitcoin console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the AMB Access Bitcoin resources in your AWS account. If you create an identity-based policy that is more restrictive than the

minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the AMB Access Bitcoin console, also attach the AMB Access Bitcoin *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Accessing Bitcoin networks

Note
In order to access the public endpoints for the Bitcoin `mainnet` and `testnet` to make JSON-RPC calls, you will need user credentials (`AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`) that have the appropriate IAM permissions for AMB Access Bitcoin.

Example IAM Policy to access all Bitcoin Networks

This example grants an IAM user in your AWS account access to all the Bitcoin networks.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessAllBitcoinNetworks",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:InvokeRpcBitcoin*"
            ],
            "Resource": "*"
        }
    ]
}
```

Example IAM Policy to access the Bitcoin Testnet network

This example grants an IAM user in your AWS account access to the Bitcoin `testnet` network.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessBitcoinTestnet",
            "Effect": "Allow",
            "Action": [
                "managedblockchain:InvokeRpcBitcoinTestnet"
            ],
            "Resource": "*"
        }
    ]
}
```

Troubleshooting Amazon Managed Blockchain (AMB) Access Bitcoin identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AMB Access Bitcoin and IAM.

Topics

I am not authorized to perform an action in AMB Access Bitcoin

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `managedblockchain::`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 managedblockchain::GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `managedblockchain::GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AMB Access Bitcoin.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AMB Access Bitcoin. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AMB Access Bitcoin resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AMB Access Bitcoin supports these features, see How Amazon Managed Blockchain (AMB) Access Bitcoin works with IAM (p. 22).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging Amazon Managed Blockchain (AMB) Access Bitcoin events by using AWS CloudTrail

Note
Amazon Managed Blockchain (AMB) Access Bitcoin doesn't support management events.

Amazon Managed Blockchain is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Managed Blockchain. CloudTrail captures who invoked the AMB Access Bitcoin endpoints for Managed Blockchain as data plane events.

If you create a properly configured trail that is subscribed to receive the desired data plane events, you can receive continuous delivery of AMB Access Bitcoin-related CloudTrail events to an Amazon S3 bucket. Using the information that's collected by CloudTrail, you can determine that a request was made to one of the AMB Access Bitcoin endpoints, the IP address that the request came from, who made the request, when it was made, and other additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

AMB Access Bitcoin information in CloudTrail

CloudTrail is enabled on your AWS account when you create it. However, you must configure the data plane events to view who invoked the AMB Access Bitcoin Bitcoin endpoints.

For an ongoing record of events in your AWS account, including events for AMB Access Bitcoin, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all supported Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to analyze further and act on the event data collected in CloudTrail logs. For more information, see the following:

- Using CloudTrail to track Bitcoin JSON-RPCs (p. 32)
- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

By analyzing the CloudTrail data events, you can monitor who invoked the AMB Access Bitcoin endpoints.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or a federated user.

- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding AMB Access Bitcoin log file entries

For data plane events, a trail is a configuration that enables delivery of events as log files to a specified S3 bucket. Each CloudTrail log file contains one or more log entries that represent a single request from any source. These entries provide details about the requested action, including the date and time of the action, and any associated request parameters.

Note
CloudTrail data events in the log files aren't an ordered stack trace of the AMB Access Bitcoin API calls, so they don't appear in any specific order.

Using CloudTrail to track Bitcoin JSON-RPCs

You can use CloudTrail to track who in your account invoked the AMB Access Bitcoin endpoints and what JSON-RPC was invoked as *data events*. By default, when you create a trail, data events aren't logged. To record who invoked the AMB Access Bitcoin endpoints as CloudTrail data events, you must explicitly add the supported resources or resource types for which you want to collect activity to a trail. Amazon Managed Blockchain supports adding data events by using the AWS Management Console and AWS CLI. For more information, see Log events by using advanced selectors in the *AWS CloudTrail User Guide* .

To log data events in a trail, use the put-event-selectors operation after you create the trail. Use the `--advanced-event-selectors` option to specify the `AWS::ManagedBlockchain::Network` resource types in order to start logging data events to determine who invoked the AMB Access Bitcoin endpoints.

Example Data event log entry of all your account's AMB Access Bitcoin endpoints requests

The following example demonstrates how to use the `put-event-selectors` operation to log all your account's AMB Access Bitcoin endpoint requests for the trail `my-bitcoin-trail` in the `us-east-1` Region.

```
aws cloudtrail put-event-selectors \

--region us-east-1 \
--trail-name my-bitcoin-trail  \
--advanced-event-selectors '[{
    "Name": "Test",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::ManagedBlockchain::Network"] } ]}]'
```

After you subscribe , you can track usage in the S3 bucket that is connected to the trail specified in the previous example.

The following result shows a CloudTrail data event log entry of the information that's collected by CloudTrail. You can determine that a Bitcoin JSON-RPC request was made to one of the AMB Access Bitcoin endpoints, the IP address that the request came from, who made the request, when it was made, and other additional details.

```
{
        "eventVersion": "1.08",
        "userIdentity": {
            "type": "AssumedRole",
```

```
            "principalId": "AROA554UO62RJ7KSB7FAX:777777777777",
            "arn": "arn:aws:sts::111122223333:assumed-role/Admin/777777777777",
            "accountId": "111122223333"
        },
        "eventTime": "2023-04-12T19:00:22Z",
        "eventSource": "managedblockchain.amazonaws.com",
        "eventName": "getblock",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "111.222.333.444",
        "userAgent": "python-requests/2.28.1",
        "errorCode": "-",
        "errorMessage": "-",
        "requestParameters": {
            "jsonrpc": "2.0",
            "method": "getblock",
            "params": [],
            "id": 1
        },
        "responseElements": null,
        "requestID": "DRznHHEjIAMFSzA=",
        "eventID": "baeb232d-2c6b-46cd-992c-0e4033aace86",
        "readOnly": true,
        "resources": [{
            "type": "AWS::ManagedBlockchain::Network",
            "ARN": "arn:aws:managedblockchain:::networks/n-bitcoin-mainnet"
        }],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "111122223333",
        "eventCategory": "Data"
}
```